Mathématiques et sciences humaines
Mathematics and social sciences

165 | Printemps 2004
La théorie constructive des types

La théorie des types et les systèmes informatiques de traitement de démonstrations mathématiques

Type theory and proof processing system

Gilles Dowek

Édition électronique
URL : http://journals.openedition.org/msh/2904
DOI : 10.4000/msh.2904
ISSN : 1950-6821

Éditeur
Centre d'analyse et de mathématique sociales de l'EHESS

Édition imprimée
Date de publication : 1 mars 2004
ISSN : 0987-6936

Référence électronique
Gilles Dowek, « La théorie des types et les systèmes informatiques de traitement de démonstrations mathématiques », *Mathématiques et sciences humaines* [En ligne], 165 | Printemps 2004, mis en ligne le 22 février 2006, consulté le 23 avril 2019. URL : http://journals.openedition.org/msh/2904 ; DOI : 10.4000/msh.2904

Math. & Sci. hum. / Mathematical Social Sciences (42^e année, n° 165, *2004*, p. 13–29)

LA THÉORIE DES TYPES ET LES SYSTÈMES INFORMATIQUES DE TRAITEMENT DE DÉMONSTRATIONS MATHÉMATIQUES[1]

Gilles DOWEK [2]

RÉSUMÉ – *Depuis la fin des années soixante, on a vu apparaître plusieurs logiciels destinés à traiter des connaissances mathématiques, en particulier des démonstrations formelles. La réalisation de tels logiciels pose de nouveaux problèmes, en particulier celui de la conception de cadres logiques dans lesquels les mathématiques puissent être formalisées en fait. Cela renouvelle la problématique des fondements des mathématiques, jusque-là davantage concentrée sur la potentialité de la formalisatioon que sur son actualité. Plusieurs raisons expliquent que les concepteurs de tels logiciels choisissent bien souvent de formaliser les mathématiques en théorie des types, plutôt qu'en théorie des ensembles.*

MOTS CLÉS – Systèmes de traitement de démonstrations mathématiques, Théorie des types, Théorie des ensembles, Démonstration, Raisonnement, Calcul, Langage mathématique, Fonction.

SUMMARY – Type Theory and Proof Processing Systems
Since the end of the sixties, several computer programs allowing to process mathematical knowledge, and in particular mathematical proofs, have been designed. Building such programs raises new questions, in particular that of the conception of logical frameworks where mathematics can be formalized in practice. This is a new direction for fundational studies, more interested, so far, in formalization of mathematics in principle, than in practice. Several reasons explain that the designers of such programs often chose type theory rather than set theory to formalize mathematics.

KEYWORDS – Proof Processing Systems, Type Theory, Set Theory, Proof, Deduction, Computation, Mathematical Language, Function.

1. QU'EST-CE QU'UN SYSTÈME INFORMATIQUE DE TRAITEMENT DE DÉMONSTRATIONS MATHÉMATIQUES ?

1.1. Raisonnements et raisonnements

Le français attribue au moins deux significations au mot « raisonnement ». Selon *Le petit Robert*, par exemple, un raisonnement est d'abord *une activité de la raison*

[1]Article reçu le 09 avril 2003, révisé le 24 juillet 2003, accepté le 30 septembre 2003.

[2]Laboratoire d'informatique (LIX), École polytechnique 91128 Palaiseau cedex, Gilles.Dowek@polytechnique.fr

avant d'être *une suite de propositions liées les unes aux autres selon des principes déterminés*. L'investigation logique semble s'être fixé comme point de départ la réduction de la première acception de ce mot à la seconde, c'est-à-dire l'idée que nombre des jugements de vérité auxquels nous accédons par l'activité de notre raison sont également accessibles par la construction d'une suite de propositions liées les unes aux autres selon des principes déterminés. On peut, sans doutes, faire remonter à l'Antiquité – Aristote, les stoïciens, mais aussi Euclide – ce projet de réifier les pensées, ou si l'on préfère de les expulser hors de la conscience.

Ce projet a suscité au cours de l'histoire beaucoup de réticences, mais aussi beaucoup d'enthousiasme. Les bénéfices de cette réification sont en effet nombreux : elle rend les pensées communicables, elle soulage nos mémoires en permettant la constitution d'archives matérielles, elle permet de mettre en évidence des erreurs de raisonnement et donc de résoudre, au moins partiellement, certains désaccords. Enfin, elle renseigne sur les processus mentaux du raisonnement que nous supposons, peut-être hâtivement[3], homologues aux règles de déduction, c'est-à-dire aux principes qui, dans un raisonnement, lient les propositions les unes aux autres.

En poursuivant ce travail de déshumanisation, on aboutit à l'idée que non seulement les démonstrations peuvent être expulsées hors de la conscience, mais que, de plus, beaucoup d'opérations que nous effectuons sur ces démonstrations, par exemple la vérification de leur correction, peuvent être, elles aussi, expulsées hors de la conscience, puisque qu'elles ne demandent que l'exécution d'un algorithme, et souvent d'un algorithme assez simple. Il est donc non seulement possible de stocker des démonstrations hors de la conscience, sur des supports matériels comme des livres, des bandes magnétiques ou des disques d'ordinateurs, en vue d'une restitution future, mais il est également possible que ces supports matériels effectuent certaines opérations sur ces démonstrations, par exemple distinguent les démonstrations correctes des démonstrations incorrectes. On aboutit ainsi au projet de fabriquer des *systèmes informatiques de traitement de démonstrations mathématiques*.

Il est toujours difficile de dater précisément l'apparition d'une idée. On peut, par exemple, dire que cette idée était implicite chez G.W. Leibniz, voire chez R. Lulle, ou qu'elle était présente dès les premières tentatives, à la fin des années cinquante, d'écrire des programmes capables de démontrer des théorèmes simples. Il est sans doute plus exact de la dater de 1967, quand a débuté la conception du programme *Automath*[4], par N.G. De Bruijn et son équipe.

1.2. Quels traitements ?

Un système de traitement de démonstrations mathématiques est donc un programme informatique capable d'effectuer un certain nombre d'opérations sur des démonstrations mathématiques. La première, et la plus importante, de ces opérations est la

[3] J.B. Joinet, "Proofs, reasonning and the metamorphosis of logic", [à paraître en 2004].

[4] N.G. de Bruijn, "A survey of the project Automath", *To H.B. Curry : Essays in combinatory logic, lambda calculus and formalism*, J.P. Seldin and J.R. Hindley (eds.), Academic Press, 1980, p. 579-606. Reprinted in : *Selected papers on Automath*, "Studies in Logic", R.P. Nederpelt, J.H. Geuvers and R.C. de Vrijer (eds.), vol. 133, North-Holland, 1994, p. 141-161.

vérification de la correction des démonstrations. L'utilisateur d'un tel programme écrit une démonstration et le programme vérifie, pas à pas, que cette démonstration est correcte, c'est-à-dire que chaque proposition découle bien des précédentes par une règle de déduction. Si ce n'est pas le cas, le programme indique le point où la démonstration est déficiente et l'utilisateur doit alors corriger son texte et le resoumettre au programme, à moins que ce constat d'erreur ne l'amène à penser que son théorème lui-même est faux, par exemple s'il a oublié une hypothèse, ou qu'il ne sait finalement pas le démontrer.

Naturellement l'écriture totalement formelle d'une démonstration serait une tâche très fastidieuse si le programme n'aidait pas l'utilisateur dans sa construction. Une manière, pour l'utilisateur, de se faire aider par le programme est d'utiliser un mode interactif, dans lequel le programme et son utilisateur coopèrent en échangeant des messages. Par exemple, l'utilisateur choisit une règle de déduction, ou une règle dérivée, pour démontrer un théorème et le programme calcule les prémisses qui doivent être démontrées pour que cette règle puisse être appliquée. L'utilisateur propose alors de nouvelles règles de déduction et ainsi de suite, jusqu'à ce que la démonstration soit achevée. Une autre possibilité est de laisser le programme effectuer les calculs numériques ou symboliques que contient la démonstration, voire tenter de démontrer seul quelques lemmes simples.

Une fois la démonstration vérifiée, le programme peut effectuer d'autres opérations comme son écriture dans un format propre à la communication avec une mathématicienne ou un mathématicien en chair et en os, son archivage et son indexation dans une base de données, sa transformation en une autre démonstration, par exemple en éliminant les coupures, c'est-à-dire les détours d'argumentation, ou, quand c'est possible, l'utilisation du tiers exclu, ou sa transformation en d'autres objets formels, par exemple des programmes informatiques.

1.3. Les mathématiques à l'ère industrielle

Si ce projet de construire des systèmes de traitement de démonstrations mathématiques était en germe dans les travaux des logiciens depuis l'Antiquité, certaines évolutions des mathématiques dans les dernières décennies du XX^{e} siècle lui ont donné une nouvelle signification et une nouvelle urgence.

La première de ces évolutions est la formidable croissance du corpus mathématique, c'est-à-dire du nombre de démonstrations publiées.

Une deuxième est l'évolution de la taille des démonstrations. Qu'on compare par exemple la taille de la démonstration du petit théorème de Fermat, que Fermat a lui-même démontré, à celle de la démonstration du grand théorème de Fermat, démontré par A. Wiles en 1994. La première occupe une demi-page, la seconde plusieurs centaines. Et même si on peut penser que des démonstrations plus courtes seront trouvées dans le futur, il est probable que cette démonstration ne sera jamais suffisamment courte pour tenir dans la marge d'un traité d'arithmétique de Diophante... Cette augmentation de la taille des démonstrations va de pair avec une organisation du travail en équipe. Un cas extrême est la démonstration du théorème de classification des groupes simples, achevée en 1980 par R. Solomon qui

occupe 15 000 pages réparties en 500 publications écrites par plus d'une centaine de mathématiciens.

Une troisième évolution concerne l'utilisation des ordinateurs dans la pratique mathématique. La première démonstration construite à l'aide d'un ordinateur est, sans doute, la démonstration du théorème des quatre couleurs proposée en 1976 par K. Appel et W. Haken, qui demande une étude de cas d'environ 1500 graphes. Si Appel et Haken sont venu à bout de cette étude, ce n'est naturellement pas en examinant eux-mêmes ces graphes l'un après l'autre, mais c'est en laissant un ordinateur calculer quelque 1200 heures. L'idée avait alors vécu que les mathématiciens n'avaient besoin que d'un tableau noir et d'un bout de craie, contrairement aux autres scientifiques, qui utilisaient des lunettes, des microscopes et des accélérateurs de particules depuis longtemps. La démonstration de la conjecture de Kepler proposée en 1997 par T. Hales, qui utilise des polynômes de 150 variables, est un autre exemple de démonstration au bout de laquelle il aurait été difficile d'arriver sans ordinateur.

Enfin, il faut aussi remarquer que les mathématiciens ont aujourd'hui perdu le monopole de la construction des démonstrations mathématiques. Des scientifiques d'autres disciplines et des ingénieurs construisent de plus en plus souvent eux-mêmes des démonstrations pour répondre aux questions qu'ils se posent. En particulier, la conception d'un programme informatique, comme un protocole de télécommunication, demande de démontrer un certain nombre de propriétés de ce programme, par exemple, dans le cas d'un protocole, que les données émises sont bien transmises ou qu'une situation de blocage ne peut pas se produire. Sans être réellement difficiles, ces démonstrations sont souvent délicates, en particulier parce que les programmes qu'elles concernent sont eux-mêmes des objets formels de grande taille, et que changer une seule lettre dans un tel programme fait passer la frontière qui sépare l'exactitude de l'erreur. Bien entendu, peu de mathématiciens appellent *théorèmes* ces propositions relatives à des objets très particuliers et qui n'ont pas la généralité des grands et beaux théorèmes mathématiques. Mais ce qui importe ici est que ces démonstrations, comme les démonstrations mathématiques, sont des suites de propositions liées les unes aux autres selon des principes déterminés.

On peut, si l'on aime les comparaisons, résumer ces différentes évolutions des mathématiques en disant que les mathématiques, ou une partie des mathématiques, sont entrées dans l'âge industriel. L'augmentation du nombre et de la taille des objets construits, le travail coopératif, l'utilisation d'outils, la spécialisation et la division du travail étant traditionnellement vus comme des signes de la transition entre l'artisanat et l'industrie. Cette entrée des mathématiques dans l'ère industrielle présente certaines analogies avec l'entrée de l'œuvre d'art à l'époque de sa reproduction technique, comme l'a souligné V. Danos.

Cette entrée des mathématiques dans l'ère industrielle pose un certain nombre de problèmes qui ne se posaient pas à l'époque où, en caricaturant à peine, une petite communauté de mathématiciens échangeait un petit nombre de démonstrations écrites à la main sur un petit nombre de feuillets. Le premier de ces problèmes est celui de la correction des démonstrations. Quand une démonstration comporte plusieurs milliers de pages, quand elle est répartie dans de nombreuses publications,

quand des dizaines de mathématiciens y ont contribué, quand elle a demandé l'utilisation d'outils de calcul, quand elle concerne un objet formel de plusieurs dizaines de pages, comment se convaincre qu'elle est correcte, et comment traquer les erreurs, sinon en utilisant un outil, comme un système de traitement de démonstrations ? On peut penser que si cette évolution des mathématiques, de l'artisanat à l'industrie, se poursuit, des outils seront de plus en plus nécessaires pour se convaincre que ce qu'on pense avoir démontré l'a bien été.

Un autre domaine où les systèmes de traitement de démonstrations mathématiques seront peut-être utiles est celui de l'organisation du corpus mathématique dans des bases de données. Au XX^e siècle, on a surtout archivé des publications écrites dans des systèmes de traitement de texte, ces documents étant indexés à l'aide de mots-clés. On peut penser qu'archiver les démonstrations elles-mêmes après avoir vérifié leur correction serait également utile, en particulier parce qu'une telle base de données pourrait être interrogée selon des critères beaucoup plus précis que par des mots-clés.

Cela explique que la réalisation de systèmes de traitement de démonstrations, et leur utilisation pour développer des démonstrations formelles, a souvent une motivation concrète, outre la motivation intellectuelle ou esthétique d'atteindre le stade ultime de la rigueur mathématique, celui où les démonstrations sont suffisamment précises pour être comprises par une machine.

2. FORMALISER EN PRINCIPE ET FORMALISER EN FAIT

Les premiers concepteurs de systèmes de traitement de démonstrations mathématiques se sont naturellement appuyés sur les travaux des logiciens de la fin du XIX^e siècle et du début du XX^e siècle qui, avec l'introduction des prédicats à plusieurs arguments et des quantificateurs par G. Frege et C.S. Peirce, puis la résolution des paradoxes par B. Russell et E. Zermelo, avaient enfin réussi à proposer quelques formalismes dans lesquels l'intégralité des mathématiques étaient exprimables.

Dans les années soixante, la théorie la plus populaire pour formaliser les mathématiques était sans doute la théorie des ensembles de E. Zermelo et A. Fraenkel. Cette théorie repose sur un socle constitué de la logique des prédicats du premier ordre qui définit une syntaxe pour les termes et les propositions et des règles de déduction, par exemple celles de G. Frege et D. Hilbert. Un projet relativement naturel aurait donc été de concevoir un programme capable de traiter des démonstrations écrites dans ce formalisme. L'utilisateur d'un tel programme aurait écrit ses propositions dans la logique des prédicats en utilisant les symboles de la théorie des ensembles, l'égalité et le symbole d'appartenance, puis aurait démontré ces propositions en utilisant les axiomes de Zermelo et Fraenkel et les règles de déduction de Frege et Hilbert.

Il est remarquable de ce projet n'ait jamais été réalisé tel quel et que tous les systèmes de traitement de démonstrations aient choisi des formalismes qui s'éloignent, plus ou moins, de la théorie des ensembles, par exemple des formalismes issus de différentes variantes de la théorie des types. Pour comprendre pourquoi, il faut faire un rapide retour sur les raisons qui avaient amené les logiciens de la fin du XIX^e

siècle et du début du XX^e siècle à proposer la théorie des ensembles comme langage de formalisation des mathématiques. Quels sont donc les problèmes auxquels la théorie des ensembles – de G. Frege à B. Russell et à E. Zermelo – devait apporter une réponse ?

Il y avait tout d'abord le problème, toujours plus ou moins présent dans les recherches logiques, de l'investigation des lois de la pensée à travers celle des règles de la déduction. Pour cela, il était nécessaire de proposer une théorie dans laquelle les mathématiques étaient formalisables, mais il n'était pas nécessaire de construire effectivement des démonstrations dans cette théorie : il suffisait de se convaincre que c'était possible. Une autre motivation était de montrer que l'arithmétique était analytique, et non synthétique *a priori*, c'est-à-dire qu'il était possible de tirer les fondements de l'arithmétique de principes purement logiques. Pour cela également, il était nécessaire que l'arithmétique soit formalisable dans les théories proposées, mais pas qu'elle y soit formalisée. Vint ensuite, après les premiers formalismes de Frege, et également la théorie des ensembles de G. Cantor, et l'apparition des paradoxes, la nécessité de donner des fondations cohérentes aux mathématiques en restreignant le schéma de compréhension d'une manière ou d'une autre. Ici aussi, le fait que les mathématiques soient formalisables dans les théories proposées était suffisant. Remarquons enfin que l'utilité de la théorie des ensembles dans les années qui ont suivi a surtout été d'être la théorie par rapport à laquelle on montrait l'indépendance de l'axiome du choix, de l'hypothèse du continu... Ici encore, c'était la potentialité de formalisation des mathématiques dans la théorie des ensembles qui importait.

Cette énumération montre que pour les logiciens de la fin du XIX^e siècle et du début du XX^e siècle, ce qui importait était bien plus la possibilité de la formalisation des mathématiques dans les théories qu'ils construisaient, que son actualité. Parmi ces logiciens, il y en avaient bien entendu certains, comme G. Peano ou A.N. Whitehead et B. Russell, qui, plus que d'autres, avaient la tentation de commencer un traité de mathématiques purement formel, c'est-à-dire d'écrire dans, et non uniquement sur, le formalisme qu'ils proposaient. Cependant aucun n'est allé bien loin dans ce projet, et il n'est pas très difficile de comprendre pourquoi : sans ordinateur, écrire plus de quelques pages dans un langage formel est une tâche plutôt pénible. La sanction et l'assistance d'une machine sont quasiment indispensables pour mener à bien une telle entreprise. De plus, sans ordinateur, une telle entreprise n'a pas grande utilité, car les ordinateurs sont pratiquement les seuls consommateurs possibles de démonstrations formelles, c'est-à-dire les seuls qui puissent leur appliquer un traitement utile, comme certifier leur correction, éliminer les coupures, les transformer en programmes, ...

3. LA FORMALISATION DES MATHÉMATIQUES, EN FAIT

Depuis les années soixante, le projet de réaliser des systèmes de traitement de démonstrations a donc amené de nouveaux problèmes aux logiciens.

Les logiciens de la fin du XIX^e siècle et du début du XX^e siècle ont proposé des formalismes dans lesquels les mathématiques étaient exprimables en principe ; ce qui importe désormais, c'est de proposer des formalismes dans lesquels les ma-

thématiques sont exprimables en fait. Puisque la théorie des ensembles n'a pas été conçue pour répondre à ce besoin, il n'est pas très surprenant qu'elle n'y réponde pas parfaitement.

3.1. Le langage des termes

La première raison pour laquelle la théorie des ensembles n'est pas adaptée à la formalisation des mathématiques en fait, est qu'elle se formule, en général, avec des axiomes d'existence, ce qui lui permet de n'utiliser qu'un langage très réduit : le symbole d'égalité et le symbole d'appartenance.

Ainsi, l'axiome de l'ensemble des parties, qui exprime que pour chaque ensemble A, il existe un ensemble B dont les éléments sont les parties de A, se formule ainsi

$$\forall A \exists B \forall C \ (C \in B \Leftrightarrow C \subseteq A)$$

où la proposition $C \subseteq A$ est une abréviation pour $\forall D \ (D \in C \Rightarrow D \in A)$.

Quand on utilise la théorie des ensembles informellement, on ajoute souvent que l'ensemble B se note $\wp(A)$. Mais *stricto sensu*, le symbole $\wp$ ne fait pas partie de la théorie des ensembles. Bien entendu, rien n'empêche d'ajouter le symbole $\wp$ au langage de la théorie des ensembles et de reformuler cet axiome

$$\forall A \forall C \ (C \in \wp(A) \Leftrightarrow C \subseteq A)$$

Le théorème de Skolem nous assure alors que cette nouvelle théorie est une extension conservatrice de l'ancienne, c'est-à-dire que les deux théories démontrent les mêmes propositions exprimées dans le langage de l'ancienne théorie.

Cependant, skolémiser ainsi les axiomes de la théorie des ensembles n'est pas suffisant, car la théorie des ensembles de Zermelo et Fraenkel contient un nombre infini d'axiomes. Pour chaque proposition P du langage originel, on pose un axiome de compréhension restreint - ou de séparation - selon lequel pour tout ensemble A il existe un ensemble B qui contient les éléments de A qui vérifient la propriété P

$$\forall A \exists B \forall C \ (C \in B \Leftrightarrow (C \in A \text{ et } P))$$

Skolémiser ce schéma d'axiome amène à introduire une infinité de symboles, indicés par les propositions du langage originel, et le langage ainsi obtenu est loin d'être facile à utiliser en fait.

On peut également utiliser des formulation alternatives de la théorie des ensembles, comme celle proposée par J. Von Neumann, P. Bernays et K. Gödel[5] qui n'a qu'un nombre fini d'axiomes, mais ici encore, le langage obtenu s'écarte, de manière importante, du langage usuel des mathématiques.

Le vernaculaire mathématique note un tel ensemble défini en compréhension $\{x \in A \mid P\}$, et c'est cette notation qui est la bonne. Cette notation présente cependant deux particularités : la première est que le terme – qui exprime une chose –

[5]K. Gödel, *The consistency of the axiom of choice and of the generalized continuum hypothesis with the axioms of set theory*, Princeton University Press, 1940. E. Mendelson, *Introduction to mathematical logic*, 3rd ed. Wadsworth, 1987.

$\{x \in A \mid P\}$ se construit à partir d'un terme A et d'une proposition – qui exprime un fait – P ; la seconde est que la variable x qui était libre dans la proposition P est désormais liée dans cette expression, autrement dit que le symbole $\{ \mid \}$, comme un quantificateur, lie une variable.

La logique des prédicats ne permet pas d'appliquer un symbole de fonction à une proposition, ni de lier une variable autrement qu'avec un quantificateur. Cela montre que la logique des prédicats est un cadre un peu étroit pour le langage mathématique. Le formalisme à étendre n'est donc pas uniquement celui de la théorie des ensembles, mais également le socle sur lequel cette théorie repose : la logique des prédicats.

Bien entendu, on peut contourner ces problèmes en introduisant des contenus propositionnels dans le langage des termes, ou en utilisant, comme H.B. Curry le propose, des combinateurs[6] plutôt que des variables liées – ce qui est l'essence de la formulation de Von Neumann, Bernays et Gödel – ou de manière plus moderne, des indices de De Bruijn[7] et des substitutions explicites[8], mais, encore une fois, tout cela a un coût élevé quand on utilise un tel langage en fait.

La construction de systèmes de traitement de démonstrations mathématiques a donc amené à poser le problème de la définition du langage des objets mathématiques, en particulier celui du statut des variables liées et de l'imbrication des termes et des propositions, problèmes quelque peu négligés par la logique traditionnelle.

3.2. Les fonctions

En poursuivant la critique du formalisme que la théorie des ensembles propose pour exprimer les mathématiques, on peut se pencher sur la manière dont cette théorie articule les notions d'ensemble et de fonction.

En théorie des ensembles, une fonction est un ensemble de couples antécédent-image, par exemple la fonction qui a un nombre entier associe son carré est l'ensemble contenant les couples $(0,0), (1,1), (2,4), (3,9), \ldots$ Dans un langage qui permet de définir un ensemble en compréhension par un terme contenant des variables liées, cet ensemble peut s'écrire $\{z \in \mathbb{N}^2 \mid \exists x \exists y\ (z = (x,y) \wedge y = x \times x)\}$. Bien entendu, rien n'empêche d'écrire également cet ensemble $x \in \mathbb{N} \mapsto x \times x$ en introduisant une nouvelle notation $\mapsto$ qui, elle aussi, lie une variable – diverses variantes notationnelles sont possibles, comme $\hat{x} \in \mathbb{N} \mapsto x \times x$ ou $\lambda x \in \mathbb{N} \mapsto x \times x$. Il n'y a donc pas de difficulté à introduire une notation explicite pour les fonctions.

L'introduction d'une notation pour l'application d'une fonction à un argument est, en revanche, plus difficile. Si f est une fonction et t un objet quelconque, le vernaculaire mathématique note ft, ou $f(t)$, l'objet u, unique s'il existe, image de t par la fonction f, c'est-à-dire tel que le couple (t,u) appartienne à la fonction f.

[6] H.B. Curry and R. Feys, *Combinatory Logic I*, North-Holland, 1958. J.R. Hindley and J. Seldin, *Introduction to combinators and lambda-calculus*, Cambridge University Press, 1986.

[7] N. G. de Bruijn, "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem", *Indagationes mathematicae*, 34, 1972, p. 381-392.

[8] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Lévy, "Explicit substitutions", *Journal of functional programming*, 1, 4, 1991, p. 375-416.

Pour rester dans le cadre syntaxique de la logique des prédicats, on peut noter cet objet $\alpha(f,t)$ en introduisant un symbole de fonction α pour l'application.

La difficulté est que, en mathématiques, les fonctions ont ordinairement un domaine de définition et qu'introduire une telle notation pour l'application d'une fonction oblige à s'interroger sur la signification de l'expression ft quand le terme t n'est pas dans le domaine de définition de la fonction f. Ici deux attitudes sont possibles et le discours sur les mathématiques oscille souvent entre l'une et l'autre. La première attitude est de prohiber la formation d'un tel terme. Ainsi, on peut dire que les expressions $1/0$ ou $\sqrt{-1}$ sont simplement prohibées par la grammaire du langage mathématique. Une deuxième attitude consiste à exploiter les ressources non de la grammaire, mais des axiomes, et à autoriser les expressions $1/0$ et $\sqrt{-1}$ en décrétant qu'elles désignent un objet mathématique conventionnel – 0, l'ensemble vide, ... – ou un objet qui n'est pas spécifié par les axiomes – et donc qu'elles peuvent désigner tel ou tel objet dans différentes extensions de la théorie. Ainsi l'expression $0 \times (1/0)$ est permise par la grammaire, mais il est impossible de démontrer que ce nombre est égal à 1 car l'axiome, ou le théorème, $x \times (1/x) = 1$ est restreint par la condition $x \neq 0$.

Ces deux notations de construction et d'application d'une fonction introduites, on peut poser un axiome - l'axiome β du lambda-calcul – qui exprime leur signification

$$(x \in A \mapsto t)\ u = [u/x]t$$

où $[u/x]t$ désigne le terme t dans lequel on a substitué les occurrences libres de la variable x par le terme u. Cet axiome exprime, par exemple, que le nombre obtenu en appliquant la fonction $x \in \mathbb{N} \mapsto x \times x$ au nombre 4 est égal à 4×4.

Bien entendu, si l'application de la fonction $x \in A \mapsto t$ à un objet situé hors de son domaine de définition A n'est pas prohibée par la grammaire, cet axiome doit être restreint par la condition $u \in A$.

La restriction de la grammaire et celle de l'axiome β ont l'une et l'autre des inconvénients. Une tentation pour éviter ces restrictions est d'étendre la fonction $x \in A \mapsto t$ à l'univers entier, et donc d'abandonner la notion de domaine de définition, en posant que, pour tout u, la valeur de l'application de la fonction $x \mapsto t$ en u est par définition égale à $[u/x]t$. Une telle définition est malheureusement circulaire. Par exemple, en prenant pour t le terme $g(xx)$ et pour u la fonction $x \mapsto (g(xx))$, on obtient que la valeur de la fonction $x \mapsto (g(xx))$ en elle-même est par définition égale à $g((x \mapsto (g(xx)))(x \mapsto (g(xx))))$. Comme on le voit le terme $(x \mapsto (g(xx)))(x \mapsto (g(xx)))$ qu'on cherche à définir apparaît dans sa propre définition. Une astuce pour éviter cette circularité consiste à remplacer les définitions par des axiomes. Cela amène à poser l'axiome β dans toute sa généralité en oubliant la notion de domaine de définition.

Comme l'a remarqué Curry, dans une telle théorie, toute fonction g a un point fixe $(x \mapsto (g(xx)))(x \mapsto (g(xx)))$ puisque une instance de l'axiome β est

$$(x \mapsto (g(xx)))(x \mapsto (g(xx))) = g((x \mapsto (g(xx)))(x \mapsto (g(xx))))$$

Cela implique par exemple qu'il n'y a pas de fonction qui prenne la valeur 0 sur l'univers entier sauf en 0. Curry a poussé assez loin l'investigation d'une axiomatisation

des mathématiques dans laquelle les fonctions n'ont pas de domaine de définition, toutes les fonctions ont un point fixe et dans laquelle il n'y a pas de fonction qui prenne la valeur 0 sur l'univers entier sauf en 0. Une telle axiomatisation n'est pas incohérente, mais elle le devient si on prend le risque d'ajouter un axiome aussi anodin en apparence que

$$\exists g \forall x \ (gx = 0 \Leftrightarrow x \neq 0)$$

Malheureusement, on est amené à poser un axiome tel que celui-ci quand on tente d'axiomatiser la négation sur les contenus propositionnels, en considérant certains objets comme le contenu propositionnel de propositions vraies et tous les autres comme le contenu propositionnel de propositions fausses.

Quand la fonction g est la négation sur les contenus propositionnels, la fonction $x \mapsto (g(xx))$ est la fonction caractéristique de l'ensemble des ensembles qui ne se contiennent pas eux-mêmes, et ce paradoxe apparaît donc comme une variante du paradoxe de Russell. Autrement dit, quand on abandonne la notion de domaine de définition, et qu'on suppose que tous les objets sont des contenus propositionnels, l'axiome β devient une variante du schéma ce compréhension non restreint de la théorie naïve des ensembles.

On est donc obligé, si on veut abandonner la notion de domaine de définition, d'introduire une distinction entre les objets qui peuvent être un contenu propositionnel et ceux, tel le point fixe de la négation, qui ne le peuvent pas. Abandonner la notion de domaine de définition introduit donc une certaine complexité dans le formalisme, qui peut être jugée excessive.

La piste consistant à ajouter la condition $u \in A$ à l'axiome β

$$u \in A \Rightarrow ((x \in A \mapsto t) \ u = [u/x]t)$$

pose un certain nombre de problèmes que nous verrons plus loin, quand nous discuterons de la transformation de certains axiomes en règles de calcul.

La dernière piste est de s'en remettre à la grammaire pour prohiber l'application d'une fonction hors de son domaine de définition. Cette piste aussi pose un certain nombre de problèmes. Le principal est que les jugements de grammaticalité et de vérité deviennent interdépendants : l'expression $(x \in A \mapsto t) \ u$ est bien formée si la proposition $u \in A$ est vraie. Pour peu qu'on puisse prendre pour A l'ensemble des machines de Turing qui terminent, la grammaticalité d'une expression n'est même plus décidable.

Une solution consiste à ajouter un troisième argument à l'application d'une fonction à son argument qui est une démonstration du fait que l'argument est dans le domaine de définition de la fonction, mais cette solution aussi est assez difficile à mettre en œuvre. D'une part, elle demande que les démonstrations soient, d'une manière ou d'une autre, des objets de la théorie, et de plus que dans la définition de ce qu'est une démonstration, on n'utilise pas la notion d'application d'une fonction, ce qui mènerait à une circularité. D'autre part, utiliser un tel formalisme en pratique demande de construire de nombreuses démonstrations, souvent évidentes, à chaque fois qu'on applique une fonction à son argument. Il est sans cesse nécessaire de rappeler que la somme de deux nombres entiers est bien un nombre entier...

Même s'il n'est pas exclu que cette voie mène un jour à des formalismes intéressants, ce n'est pas aujourd'hui la voie la plus explorée. La solution habituelle à ce problème est celle de se reposer sur une classification habituelle des propriétés d'un objet en propriétés « essentielles »et « accidentelles ». Ainsi la fonction $x \in \mathbb{R} \mapsto e^x$ a une propriété essentielle unique qui est d'être une fonction des nombres réels dans les nombres réels, et de nombreuses propriétés accidentelles : être croissante, positive, ...

La pertinence de cette classification est l'objet de nombreux débats. En faveur de cette distinction on peut avancer que les langues naturelles expriment les propriétés essentielles avec des noms communs et les propriétés accidentelles avec des adjectifs. Argument auquel on aura vite fait de répondre que cette distinction entre noms communs et adjectifs est une des plus fragiles de la grammaire traditionnelle et que, de plus, un objet a souvent plusieurs propriétés exprimables par un nom commun. Contre cette distinction, en revanche, on peut opposer que selon les situations, on veut regarder le nombre 1 comme un nombre entier, rationnel, réel, complexe, ... et donc qu'aucune de ces propriétés ne peut prétendre être « la »propriété essentielle de ce nombre. On répondra que le nombre rationnel 1 et le nombre réel 1 sont précisément des objets différents et que dans la construction des réels de Dedekind ou de Cauchy, l'ensemble des nombres rationnels n'est pas un sous-ensemble de celui des nombres réels, mais qu'il est seulement isomorphe à un tel sous-ensemble. On peut également opposer à cette distinction le fait que la théorie des ensembles doit une partie de sa simplicité conceptuelle au fait qu'elle évite cette distinction. Argument auquel on peut répondre que les utilisations pratiques de la théorie des ensembles, comme les *Éléments de mathématique* de Bourbaki, distinguent les « ensembles »et les « ensembles de base », et utilisent des lettres d'alphabets différents pour les nombres, les fonctions, les opérateurs...

Quoi qu'il en soit, en restreignant chaque objet à avoir une propriété essentielle unique, qu'on appelle son *type*, en imposant que le type d'un terme clos – c'est-à-dire de l'objet exprimé par ce terme – puisse être calculé à partir de ce terme, que le type d'un terme ouvert puisse être calculé à partir du type de ses variables, que le type d'une fonction permette de calculer son domaine de définition, qui est lui-même un type, on résout le problème de la circularité entre les jugements de grammaticalité et les jugements de vérité. Pour savoir si un terme de la forme ft est grammaticalement bien formé, il suffit de calculer le type de f et celui de t et de vérifier que le type de t est bien identique au domaine de f.

La notion de type, telle qu'on la trouve dans les *Principia mathematica* a été introduite dans le but d'éviter le paradoxe de Russell dans sa forme ensembliste, ce n'est que plus tard, en particulier lors de l'introduction de la théorie des types de A. Church[9] qu'elle a été utilisée pour résoudre ce problème du domaine de définition des fonctions.

[9] A. Church, "A formulation of the simple theory of types", *Journal of symbolic logic*, 5, 1940, p. 56-68. P. B. Andrews, *An introduction to mathematical logic and type theory : to truth through proof*, 2nd ed., Kluwer, 2002.

3.3. Raisonnement et calcul

Poursuivons notre critique de la théorie des ensembles, toujours en nous plaçant du point de vue de son utilisation pour formaliser les mathématiques en fait. Selon la méthode axiomatique, un raisonnement est une suite de propositions produites les unes à partir des autres, à l'aide de règles de déduction. Dans un tel raisonnement, on peut, en outre, tenir pour vraies certaines propositions, les axiomes, qui expriment la signification des symboles du langage, dont ils sont des définitions implicites.

Cette conception de la notion de démonstration occulte une distinction importante en mathématiques, et dans d'autres branches du savoir, celle qui oppose le raisonnement au calcul. Par exemple, comme l'avait déjà remarqué H. Poincaré, pour établir la vérité de la proposition $2 + 2 = 4$, il n'est nullement nécessaire de faire une démonstration, mais un simple calcul suffit. De plus, l'addition se définit beaucoup plus naturellement par un algorithme, par exemple par l'algorithme

$$0 + y \longrightarrow y$$

$$S(x) + y \longrightarrow S(x + y)$$

que par des axiomes.

Bien entendu, il est possible, comme on le fait par exemple dans l'arithmétique de Peano, de transformer chacune de ces règles de calcul en un axiome

$$\forall y\ (0 + y = y)$$

$$\forall x \forall y\ (S(x) + y = S(x + y))$$

et de démontrer la proposition $2 + 2 = 4$ à partir de ces axiomes en simulant chaque étape de calcul par une ou plusieurs étapes de raisonnement, mais cela revient à gommer la distinction entre raisonnement et calcul.

L'histoire des mathématiques a donné une certaine importance à cette notion de calcul : qu'on pense, pour ne citer que quelques exemples, aux travaux des mathématiciens médiévaux sur la conception d'algorithmes pour les opérations arithmétiques élémentaires sur les nombres en notation décimale, ou au développement du calcul intégral à l'époque moderne qui permet de substituer des calculs aux raisonnements archimédiens. Cette notion de calcul joue également un rôle important dans l'histoire de la logique, à travers, par exemple, la théorie de la calculabilité, ou celle des démonstrations constructives. Il peut paraître de ce fait surprenant que la méthode axiomatique néglige cette notion à ce point et fasse du calcul un simple cas particulier du raisonnement.

Encore une fois, on comprend pourquoi si on se souvient que le but des logiciens de la fin du XIX^e siècle et du début du XX^e siècle était de proposer des formalismes dans lesquels les mathématiques puissent être exprimées en principe. Dans ce cas, la différence entre raisonnement et calcul n'est pas pertinente. Elle le devient, en revanche, quand on cherche à construire des formalismes dans lesquels les mathématiques peuvent être exprimées en fait : l'utilisateur d'un système de traitement de démonstrations mathématiques comprendrait mal qu'un ordinateur ne puisse pas "démontrer" tout seul que deux et deux font quatre ou que la primitive

de $x \in \mathbb{R} \mapsto x^2$ est $x \in \mathbb{R} \mapsto x^3/3$. De même, quand on cherche à exprimer une démonstration dans un format lisible par un mathématicien, à l'archiver ou à la transmettre sur un réseau, on gagne à l'expurger des arguments calculatoires qui peuvent être reconstruits par le destinataire de cette démonstration.

Parmi les règles de calcul, une règle joue un rôle particulièrement important. C'est la règle

$$(x \in A \mapsto t)\ u \longrightarrow [u/x]t$$

qui replace l'axiome β

$$(x \in A \mapsto t)\ u = [u/x]t$$

C'est pour pouvoir transformer cet axiome en une règle de calcul qu'il est important que cet axiome ne soit pas restreint par la condition $u \in A$.

Cette règle peut mener à des calculs infinis dans un cadre où les fonctions n'ont pas de domaine de définition, puisque le terme $(x \mapsto (g(xx)))(x \mapsto (g(xx)))$ se réduit en $g((x \mapsto (g(xx)))(x \mapsto (g(xx))))$, mais elle se termine quand on se restreint aux termes bien typés.

Cette idée de transformer l'axiome β en une règle de calcul, et de raisonner avec des propositions β-réduites, en réduisant les propositions systématiquement après chaque étape de raisonnement, est plus ou moins implicite dans les travaux de Church, mais on peine cependant à trouver une formulation claire de la théorie des types de Church dans laquelle les propositions sont systématiquement β-réduites.

3.4. Les démonstrations, objets de la théorie

Achevons notre critique de la théorie des ensembles par une discussion sur le statut des démonstrations dans cette théorie.

Au cours de l'histoire, l'espace des objets mathématiques, au départ composé essentiellement des nombres entiers et des figures géométriques, n'a cessé de s'enrichir. Cet enrichissement a souvent procédé par l'incorporation d'objets considérés jusqu'alors comme périphériques à cet espace. Ainsi on utilise depuis l'Antiquité les fonctions trigonométriques pour parler du sinus ou du cosinus de tel ou tel angle, mais ces fonctions ne sont devenues des objets mathématiques à part entière qu'au XVII^e ou au XVIII^e siècle, quand on a commencé à parler non seulement des propriétés du sinus d'un angle ou d'un autre, mais des propriétés de la fonction sinus elle-même.

De manière similaire, en théorie des ensembles et en théorie des types simples, les démonstrations restent des objets périphériques. On peut, dans ces théories, exprimer le fait que 7 est un élément de l'ensemble des nombres premiers ou que l'ensemble des nombres premiers est infini, mais on ne peut pas exprimer le fait qu'un certain objet est une démonstration de cela. Bien entendu, dans ces théories, on peut toujours coder une démonstration par un nombre entier ou un ensemble héréditairement fini et exprimer un prédicat $D(n, p)$ selon lequel p est le code de Gödel d'une proposition et n est le code de Gödel d'une démonstration de cette proposition. Mais, d'une part, on est obligé de distinguer la proposition de son code de Gödel, sans pouvoir exprimer de prédicat de vérité dans le langage, et, d'autre part,

on est obligé de recourir à un codage *ad hoc* des démonstrations comme des nombres entiers, renonçant ainsi à capturer leurs propriétés mathématiques essentielles.

Même si ce point est encore l'objet de débats, on peut dire que l'utilisation d'une formalisation des mathématiques dans laquelle les démonstrations sont des objets à part entière, comme les nombres, les ensembles ou les fonctions, et peuvent donc être exprimées par des termes, facilite la réalisation d'un système de traitement de démonstrations mathématiques. En effet, quand on veut transformer une démonstration en une autre ou en un programme, l'archiver, la transmettre sur un réseau de manière à ce qu'elle soit revérifiable à l'arrivée, faire coopérer des programmes différents, ... il est nécessaire d'utiliser un langage commun pour exprimer les démonstrations et au lieu d'introduire un nouveau langage dont les expressions sont des suites de propositions, par exemple de la théorie des ensembles ou de la théorie des types simples, il est souvent plus simple d'utiliser un formalisme dans lequel les démonstrations sont des objets et peuvent donc être exprimées par un terme du langage lui-même.

De tels formalismes, dans lesquels les démonstrations sont des objets à part entière, ont commencé à être proposés à partir des années soixante. Malheureusement, on continue à appeler ces formalismes « théorie des types », alors qu'ils n'ont qu'un lointain rapport avec les théorie des types du premier type, qui vont des *Principia* à la théorie des types simples de Church. Ces théories des types du deuxièmes type, qui incluent la théorie du système *Automath*, la *Théorie des types intuitionniste* de Martin-Löf[10], le *Calcul des constructions*[11], et ses diverses extensions, s'appuient sur une représentation fonctionnelle des démonstrations que l'on doit à L.E.J. Brouwer, A. Heyting et A. Kolmogorov. Dans cette interprétation, une démonstration d'une proposition de la forme $A \Rightarrow B$ est une fonction qui associe une démonstration de B à toute démonstration de A. Ainsi, si on a une démonstration f de la proposition $A \Rightarrow B$ et une démonstration a de la proposition A, la règle de *modus ponens* nous permet d'en déduire la proposition B et la démonstration est simplement le terme fa obtenu en appliquant la fonction f à l'objet a. L'important ici est de remarquer que ce symbole d'application est le même que celui permettant d'appliquer n'importe quelle fonction à n'importe quel objet, par exemple la fonction sinus au nombre 0. Naturellement, pour obtenir une démonstration de la proposition B, on ne peut pas appliquer une démonstration de la proposition $A \Rightarrow B$ à une démonstration d'une proposition A' distincte de A, car on appliquerait une fonction à un objet qui se trouve hors de son domaine de définition.

On est donc ainsi ramené à la question précédente concernant la manière de prohiber l'application d'une fonction à un objet situé hors de son domaine de définition. Si on demande d'ajouter à l'application un troisième argument qui est une démonstration du fait que l'objet a appartient bien au domaine de f, c'est-à-dire que c'est bien une démonstration de la proposition A, on risque d'introduire une régression à l'infini. En effet, cette démonstration que a est une démonstration de A risque elle-même d'utiliser des applications qu'il va falloir justifier... De plus, savoir

[10] P. Martin-Löf, "Intutionistic type theory", *Bibliopolis*, 1984.

[11] Th. Coquand and G. Huet, "The calculus of constructions", *Information and computation* 76, 1988 p. 95-120.

qu'un certain objet est une démonstration d'une certaine proposition ne requiert pas de démonstration, au risque, encore une fois, d'une régression à l'infini, mais doit pouvoir se lire sur la face de l'objet, ou plutôt d'un terme exprimant cet objet. L'alternative, comme nous l'avons vu est de s'en remettre à la grammaire pour prohiber l'application d'une fonction hors de son domaine de définition. Cela demande d'associer à chaque proposition une propriété essentielle, un type : le type de ses démonstrations. Cette association d'un type à chaque proposition du langage est ce qu'on appelle l'isomorphisme de Curry-De Bruijn-Howard, et cela demande des langages de types plus riches que celui des types simples.

Ces langages de types plus riches étendent souvent simultanément la puissance des règles de calcul de ces formalismes.

3.5. L'influence des utilisations

On commence à voir dans quelles directions la théorie des ensembles doit être modifiée ou enrichie pour pouvoir être adaptée à la formalisation des démonstrations en fait. Il en reste une dernière.

Si beaucoup de systèmes de traitement de démonstrations sont des systèmes généralistes, développés indépendamment des utilisations qu'on compte en faire, suivant ainsi la tradition de neutralité ontologique de la logique, certains systèmes ont été développés en vue d'applications déterminées, en particulier la preuve de programmes, ou bien, tout en restant généralistes, ont subi une certaines influence des utilisations visées.

On peut expliquer l'importance que jouent dans ce domaine la notion de constructivité. Certes, l'interprétation fonctionnelle des démonstrations a d'abord été comprise dans le cadre des démonstrations constructives – comme l'évoquent les noms de Brouwer, Heyting et Kolmogorov – et ce n'est que dans la dernière décennie du XX^{e} siècle que des extensions ont été proposées pour la logique classique. Mais on peut aussi expliquer cet intérêt pour les démonstrations constructives par la volonté de transformer ces démonstrations en programmes informatiques démontrés corrects.

Cette influence des utilisations visées est plus forte encore dans un programme comme LCF[12] – *Logic for computable functions* – où l'emphase est mise dès la conception de la logique sur le fait que les fonctions représentent des programmes et sont donc des fonctions partielles qui peuvent ne pas terminer, ou ACL2[13] dont le langage des termes est le langage de programmation LISP.

4. LA THÉORIE DES TYPES

Pour résumer, on peut tenter une classification des innovations qu'on est incité à adopter pour adapter la théorie des ensembles à la formalisation des mathématiques en fait.

[12]M. Gordon, R. Milner, and C. Wadsworth, "Edinburgh LCF", *Lecture notes in computer science* 78, Springer-Verlag, 1979.

[13]`http ://www.cs.utexas.edu/users/moore/acl2/`

Sur le plan du socle syntaxique sur lequel la théorie repose, il faut enrichir la logique des prédicats en introduisant des symboles de fonction qui permettent de lier des variables dans les termes, en introduisant des symboles de fonction qui permettent d'imbriquer les termes et les propositions et en introduisant des types, ou des sortes, qui restreignent la grammaire des termes. Sur le plan de la définition des conditions de vérité des énoncés, il faut permettre de définir une théorie non seulement avec des axiomes, mais aussi avec des règles de calcul. Sur le plan de la théorie elle-même, on est incité à introduire un langage explicite pour les objets mathématiques – et non uniquement deux symboles de prédicats pour l'égalité et l'appartenance – et on est incité à prendre un langage de termes suffisamment riche pour pouvoir exprimer des fonctions et les objets obtenus en appliquant une fonction à un argument, et également des démonstrations.

Bien entendu, certains de ces points, sont encore l'objet de débats.

On peut comprendre, à présent, pourquoi de nombreux programmes de traitement de démonstrations mathématiques utilisent une forme ou une autre de la théorie des types : la théorie des types simples, ou l'une de ses variantes, pour certains d'entre eux – HOL[14], PVS[15], Isabelle[16], ... – une théorie des types dans laquelle les démonstrations sont des objets pour d'autres – Coq[17], Lego[18], Alfa[19], ... D'un point de vue sociologique, on peut noter que, ces dernières années, les travaux sur la théorie des types ont été en grande partie menés par des équipes qui développent aussi des systèmes de traitement de démonstrations, ou par des équipes qui ont des liens étroits avec de telles équipes, comme par exemple celles regroupées dans le projet européen TYPES.

En effet, la théorie des types simples de Church introduit un langage explicite pour les objets mathématiques et non uniquement des axiomes d'existence, elle introduit un symbole $\mapsto$ qui permet de lier une variable dans un terme, elle donne aux propositions le statut de termes, elle donne aux fonctions le statuts d'objets primitifs, elle introduit un symbole pour l'application d'une fonction à son argument et propose des domaines de définition simples aux fonctions, ce qui permet de s'en remettre à la grammaire pour prohiber l'application d'une fonction hors de son domaine de définition, elle permet enfin la transformation de l'axiome β en règle de calcul.

Les théories des types du deuxième type permettent, en outre, de représenter les démonstrations par des termes, ce qui permet l'archivage, la transmission, la revérification de ces démonstrations, leur transformation en des programmes, ... sans introduire de langage supplémentaire pour ces démonstrations.

[14] `http ://www.cl.cam.ac.uk/Research/HVG/HOL/`
[15] `http ://pvs.csl.sri.com/`
[16] `http ://www.cl.cam.ac.uk/Research/HVG/Isabelle/`
[17] `http ://coq.inria.fr/`
[18] `http ://www.dcs.ed.ac.uk/home/lego/`
[19] `http ://www.math.chalmers.se/~hallgren/Alfa/`

5. DU CLAIR AU SOMBRE ET DU SOMBRE AU CLAIR

L'histoire des recherches sur ces formalismes adaptés à la formalisation des mathématiques en fait illustre un schéma fréquent dans l'histoire des sciences, celui d'un passage du clair au sombre, suivi d'un processus de décantation, du sombre au clair.

Par exemple, indépendamment de toute théorie, il est possible d'étendre la logique des prédicats de manière à permettre la formation de termes qui ont des variables liées. On obtient alors un cadre général relativement simple dans lequel il est possible d'exprimer de nombreuses théories, parmi lesquelles la théorie des types. Cependant, bien qu'ils soient plus simples, ces cadre généraux ne sont apparus qu'après la théorie des types qui en est une instance.

Les nouvelles idées ne sont pas apparues dans toute la clarté de la généralité, mais dans la complexité introduite par l'interférence avec de nombreuses autres idées dans une théorie particulière, et quelque peu accidentelle. Les progrès ne se sont donc pas fait du clair au clair, mais du clair au sombre, puis du sombre au clair.

Bien souvent, on n'attend pas que les innovations aient fini de décanter pour les utiliser. Et, parmi toute la combinatoire des formalismes possibles, qu'on peut identifier une fois les idées décantées – avoir des variables liées ou non, les règles de calcul ou non, les démonstrations comme objets ou non, ... – les formalismes qui existent et qui ont déjà été étudiés ont un certain avantage, même si leur existence est parfois accidentelle.

Ainsi, pour les concepteurs de systèmes de traitement de démonstrations, le premier avantage de la théorie des types était d'être là. Aujourd'hui que les innovations apportées par la théorie des types commencent à décanter, on voit apparaître d'autres combinaisons possibles. Ainsi, parmi les recherches effectuées autour des systèmes de traitement de démonstrations mathématiques, on commence à voir réapparaître des travaux sur la théorie des ensembles, mais une théorie des ensembles transformée, avec un langage de termes, des variables liées, des règles de calcul, des termes pour les démonstrations, ...

La théorie des types continuera-t-elle à être le formalisme favori des concepteurs de systèmes de traitement de démonstrations, ou aura-t-elle été le vecteur par lequel de nombreuses innovations seront apparues, avant de devenir autonomes ? Seule l'observation attentive du futur pourra le dire.